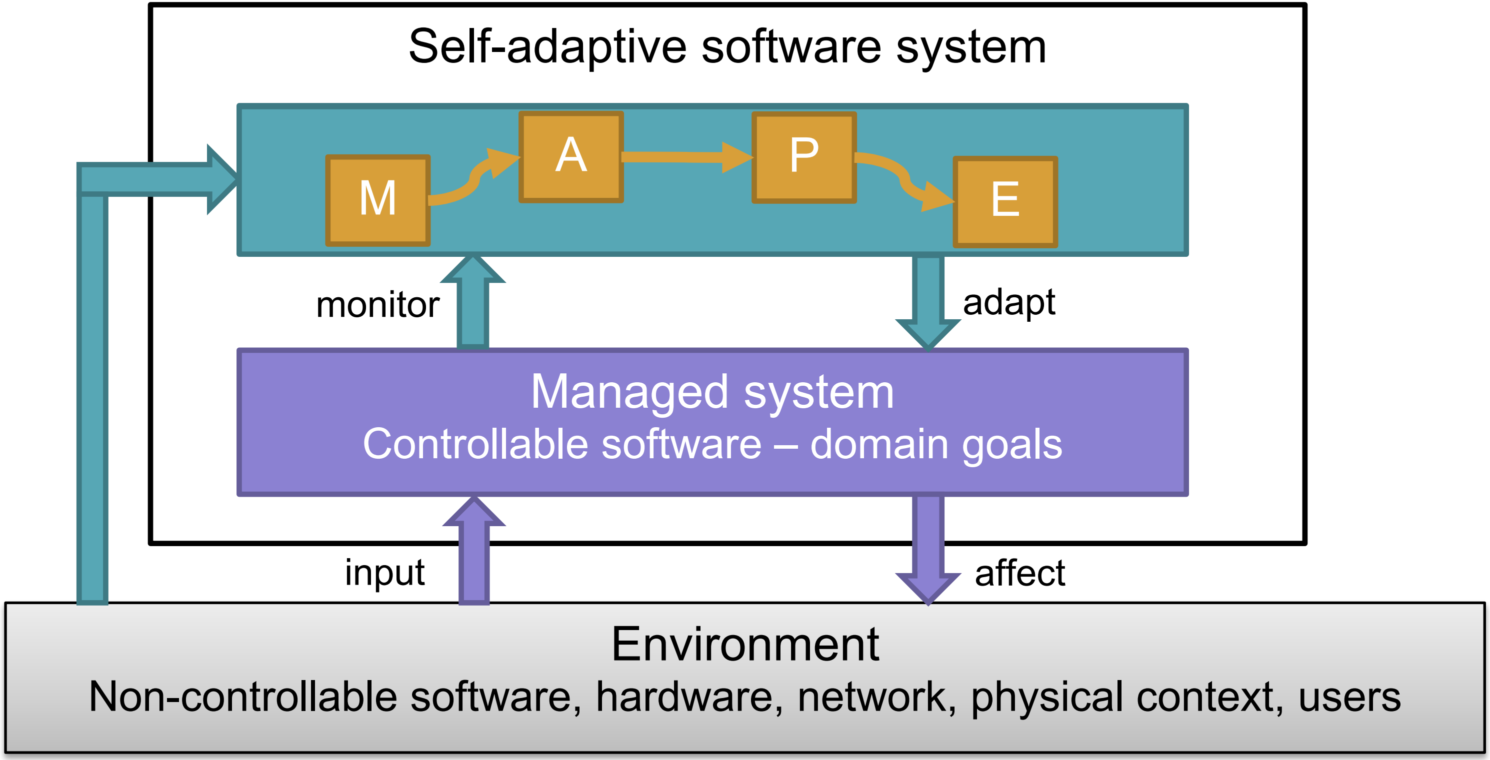
Strengthening self-adaptation in the face of unanticipated situations

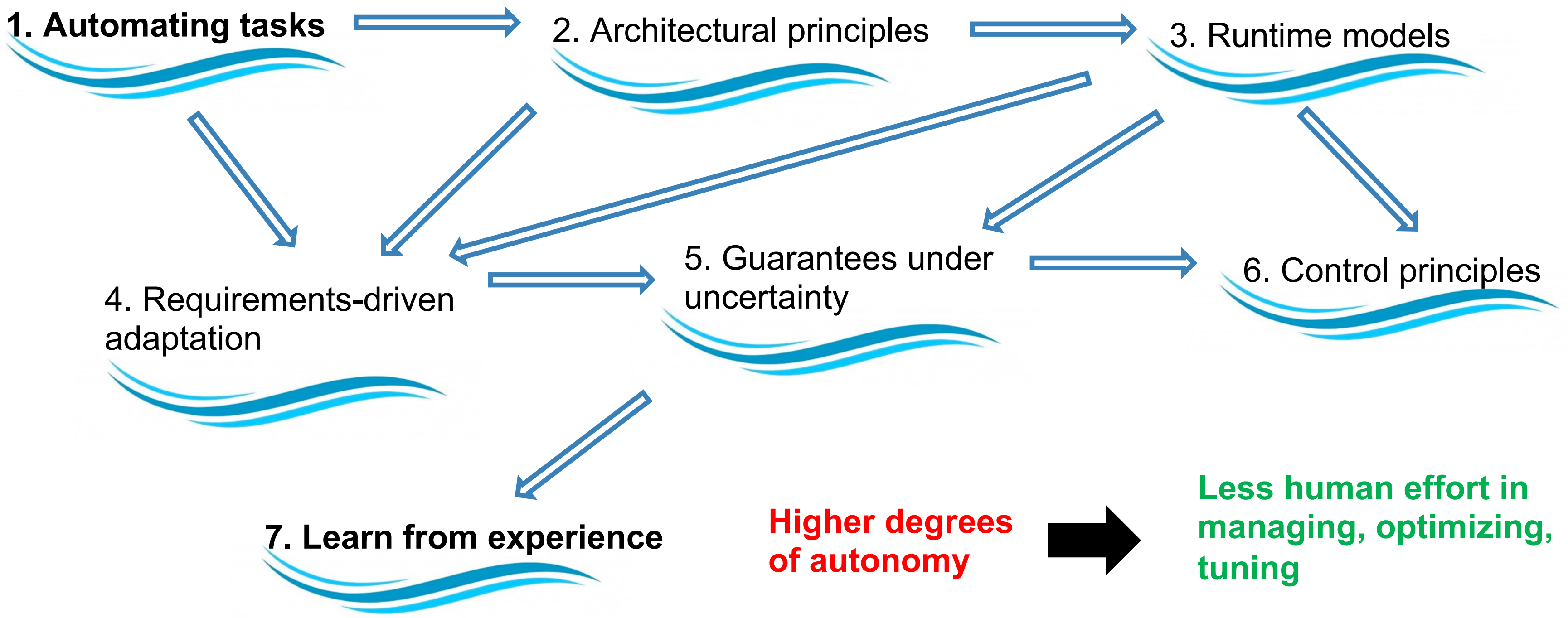
Ilias Gerostathopoulos
Assistant professor
Software and Sustainability Group
Computer Science Department
Vrije Universiteit Amsterdam
i.g.gerostathopoulos@vu.nl

CASA@ECSA 2021, 13th September 2021

A self-adaptive system view



From automation to autonomy



Danny Weyns, "An Introduction to Self-adaptive Systems: A Contemporary Software Engineering Perspective", Wiley 2020

NASA’s evolutionary stages

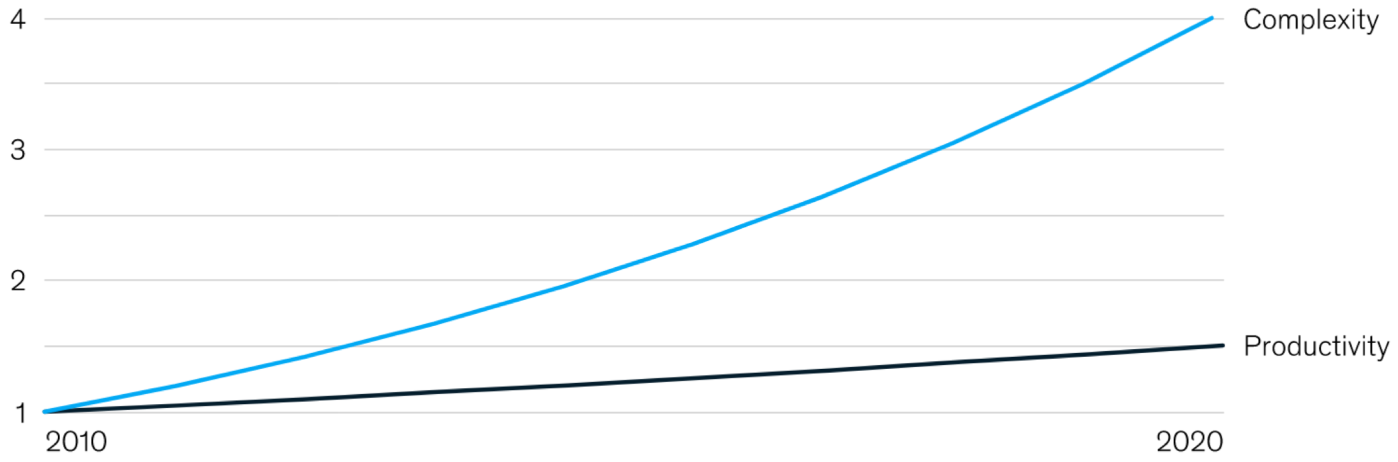
- Stage 1: “Resilient System”
  - *System performs resource management and health management functions. Executes “tactical” activity plans provided by operations team. Uses and adapts models of internal state. Control via closed-loop commanding. Adapts detailed plan to address minor anomalies.*
- Stage 2: “Independent System”
  - *System generates tactical activity plan based on science directives (“strategic plan”) provided by science team. Uses and adapts models of internal state and environment. Possible to reduce size of mission operations team.*
- Stage 3: “Self-Directed System”
  - *System develops science strategic plan and tactical plans based on high-level objectives. Responds to novelty by adjusting plans within context of objectives. Possible to reduce size of science operations team.*

From “Challenges and Opportunities in Robotic Space Exploration”, John Day, NASA
UKRI TAS Resilience Node Talk 7, April 2021, https://www.youtube.com/watch?v=yavzrbIqOkI

Complexity-productivity gap in the automotive industry

Software complexity is increasing more quickly than productivity.

Relative growth of software complexity and productivity over time, indexed for automotive features



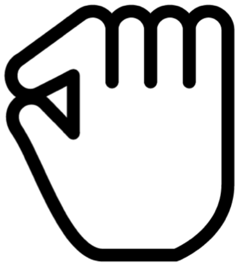
McKinsey & Company, “When code is king: Mastering automotive software excellence”, February 17, 2021

Main hypothesis

We cannot reach higher degrees of autonomy if we don’t enable systems to deal with situations not anticipated by their designers!

The rest of the talk

Two inspirational moments and the "research stories" that followed



Acknowledgement

This presentation is based on research performed in collaboration with the following great colleagues:

- Architecture homeostasis: **Tomas Bures, Frantisek Plasil, Dominik Skoda, Alessia Knauss**
- Planning as Optimization: **Erik Fredericks, Thomas Vogel, Christian Krupitzer**

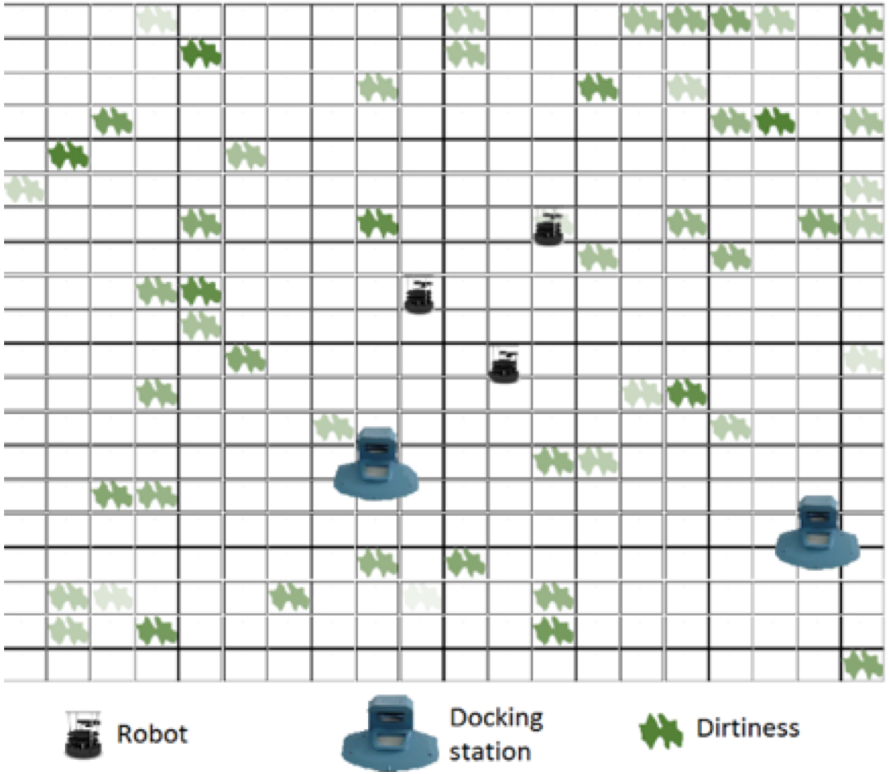
inspirational moment #1

Consider adding a **pinch of uncertainty** to your systems

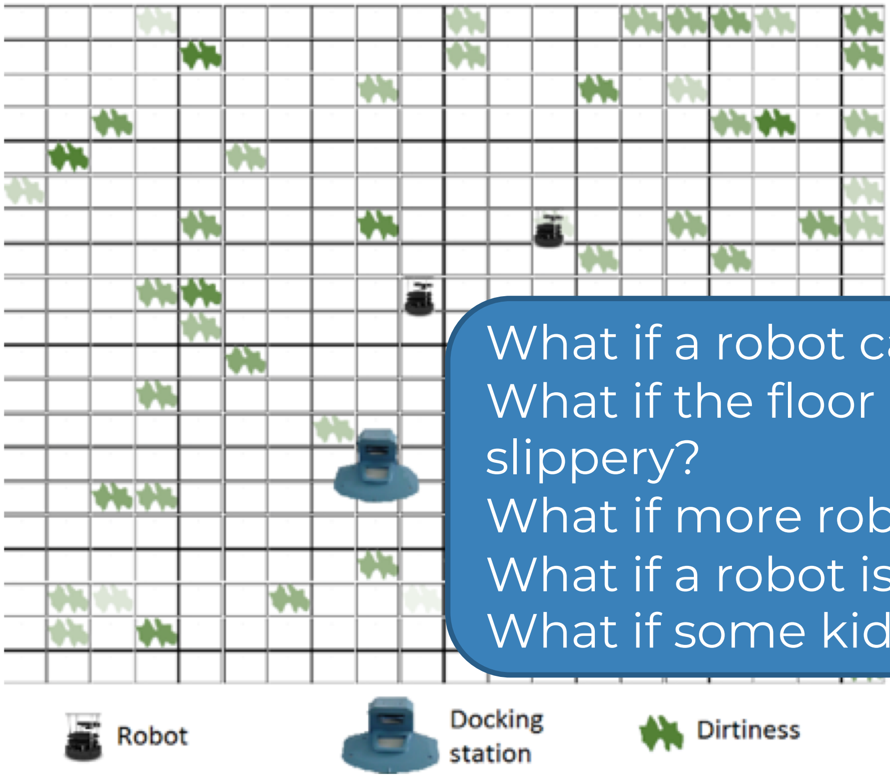
You may find that they work better!

Maarten van Steen, ECSA 2015, keynote

“Cleaning robots” system



"Cleaning robots" system



What if a robot cannot locate itself anymore?
What if the floor becomes too wet and slippery?
What if more robots join the group?
What if a robot is out of power?
What if some kids start playing with the robot?

Self-adaptation to the rescue?

Adjusting a system’s behavior and/or structure can indeed help

- Choosing a **different sensor** that provides the same values
- Choosing a **different service** with lower latency to call
- Reducing the motor speed
- …

However, designers have to anticipate all potential situations and actions!

Impractical, if not **impossible**, for many real-life systems

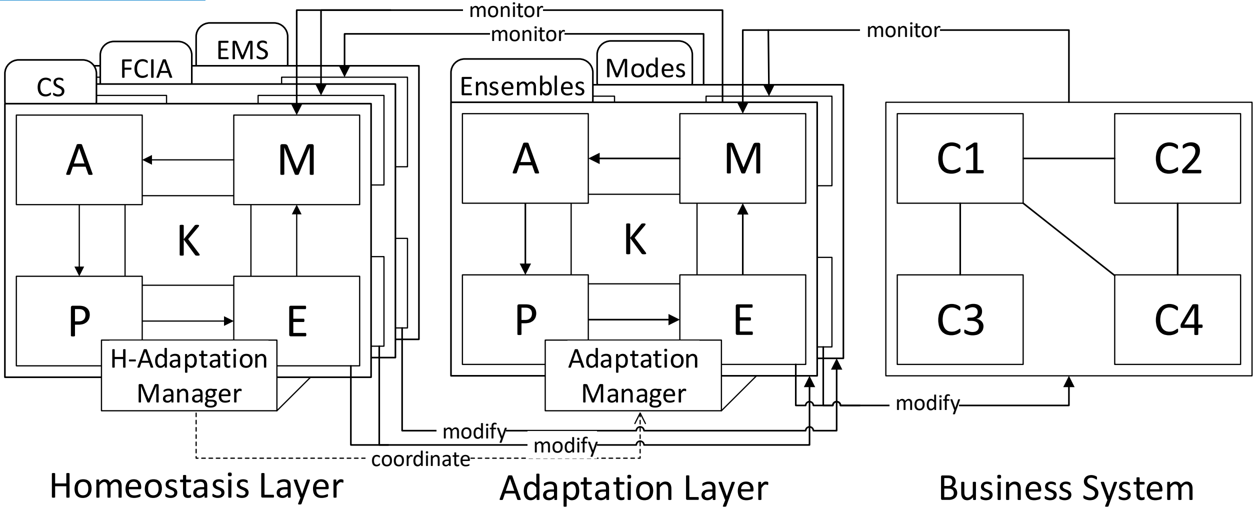
When self-adaptation is not enough...

What if Instead of trying to identify all potential situation-action pairs, we identify a number of them and then allow the actions to be **slightly changed** at runtime?

Then More situations (even unanticipated) could be handled

Finally
- The system may cope better with runtime uncertainty
- Increased **homeostasis** [ability for the system to maintain its normal operating state and implicitly repair abnormalities or deviations from expected behavior]

The big picture



Homeostasis layer introduces (a pinch of) uncertainty to the adaptation strategies

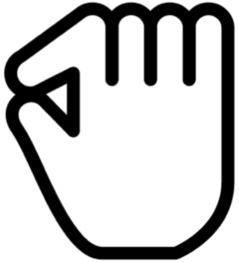
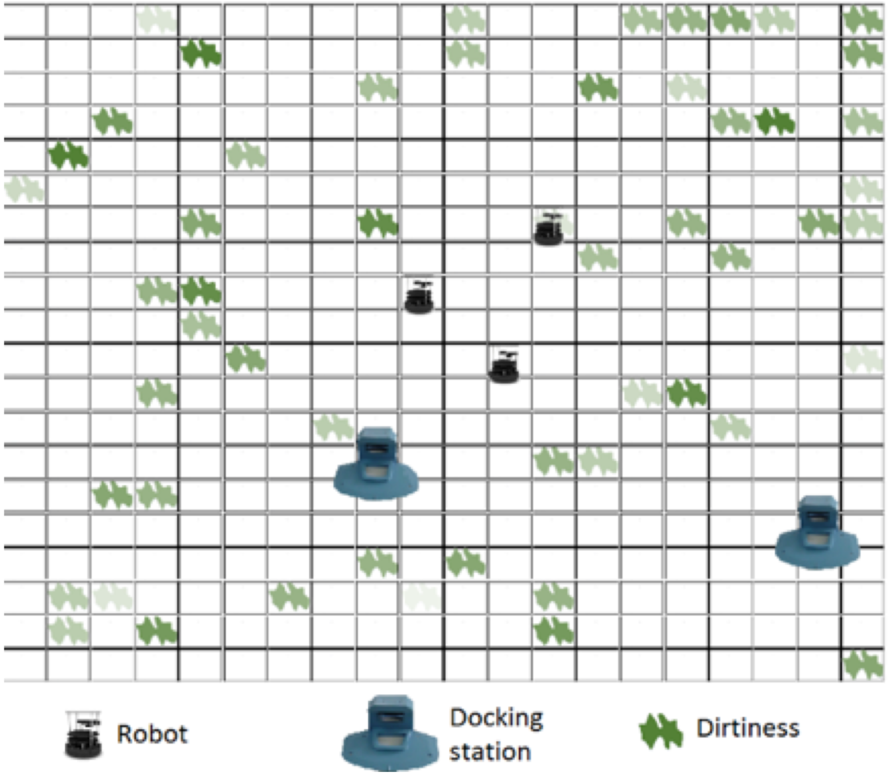
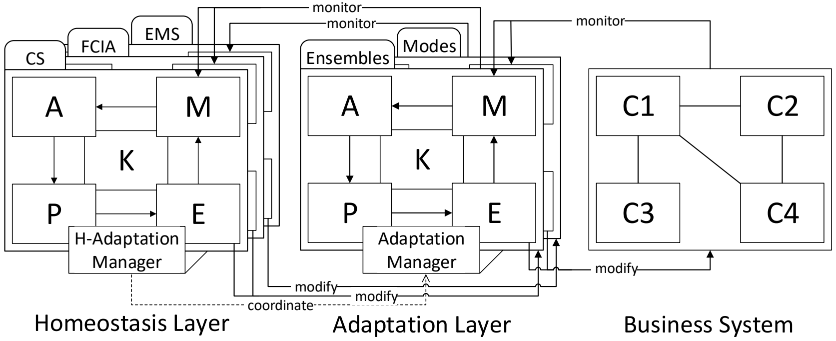


Illustration on Cleaning Robots





Adaptation layer:

2 adaptation strategies

Homeostasis layer:

3 homeostatic mechanisms

Illustration on Cleaning Robots

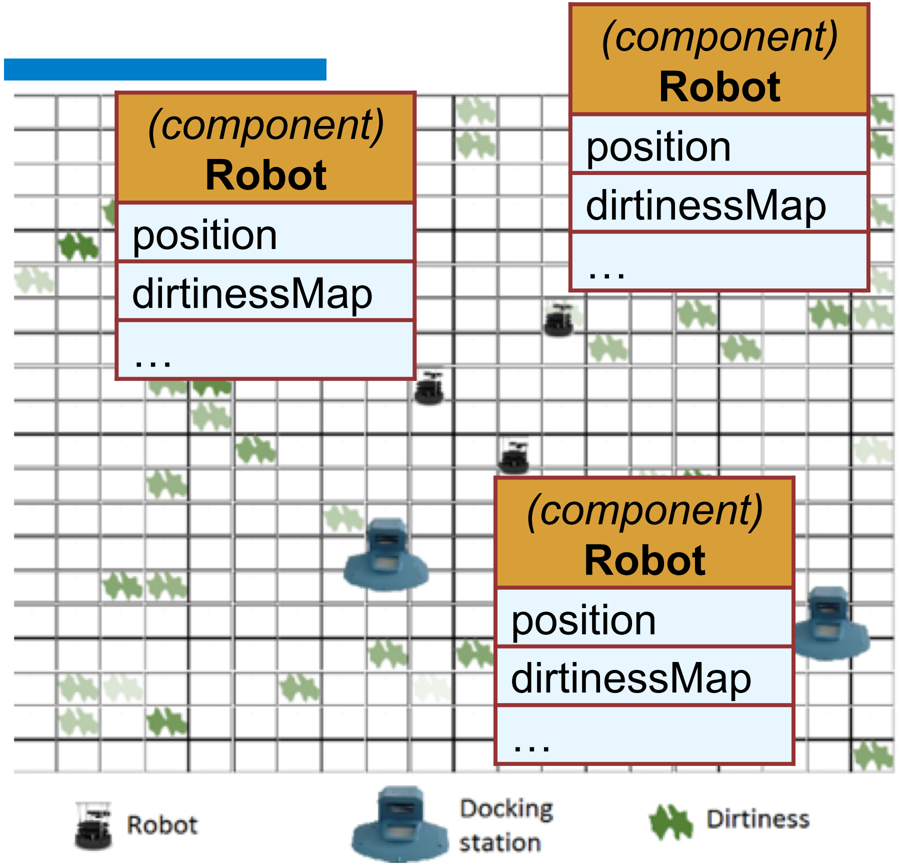


Illustration on Cleaning Robots

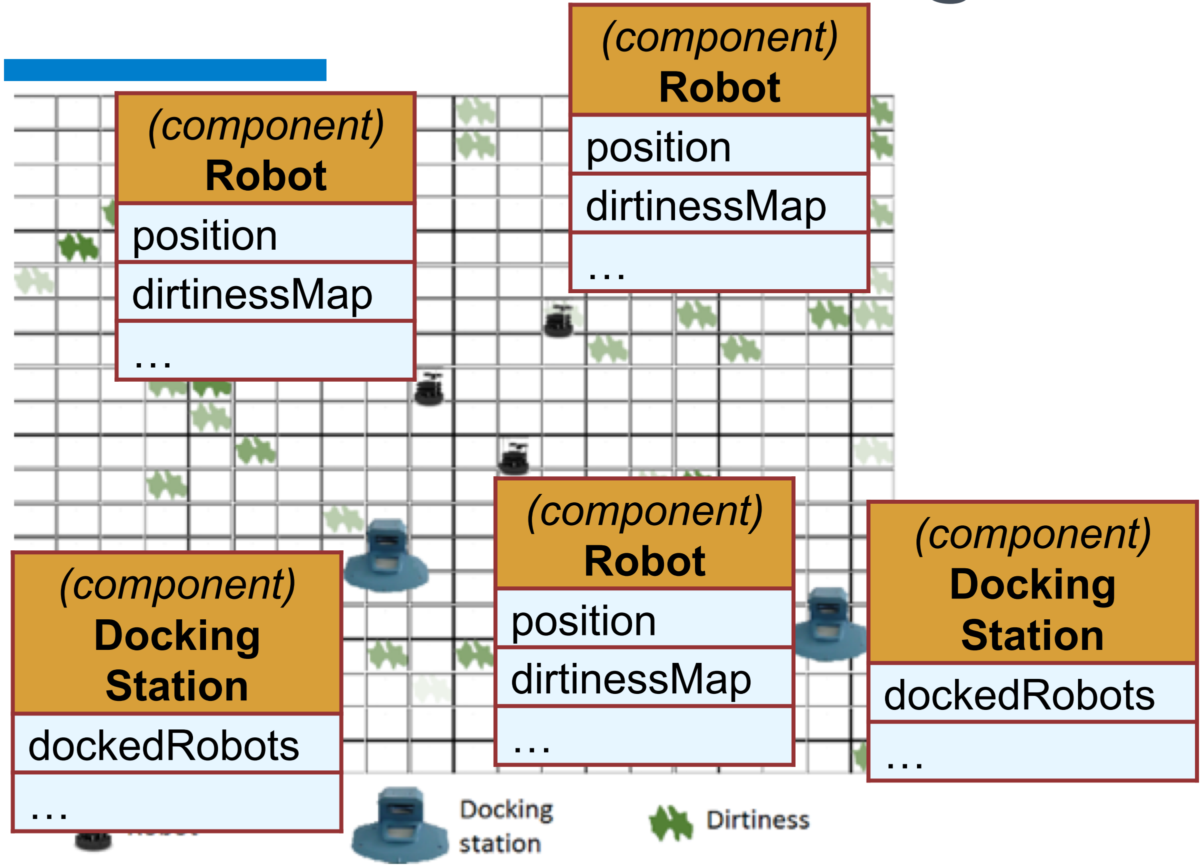
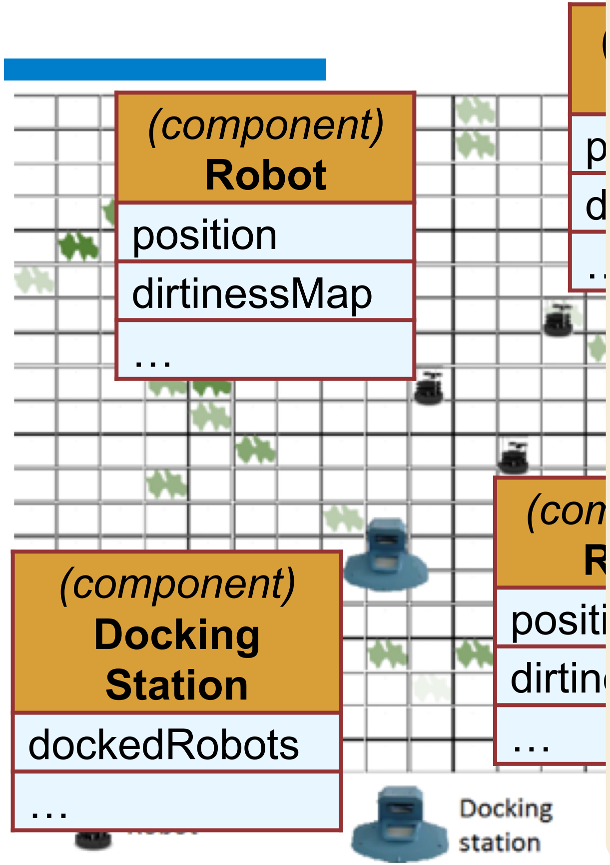
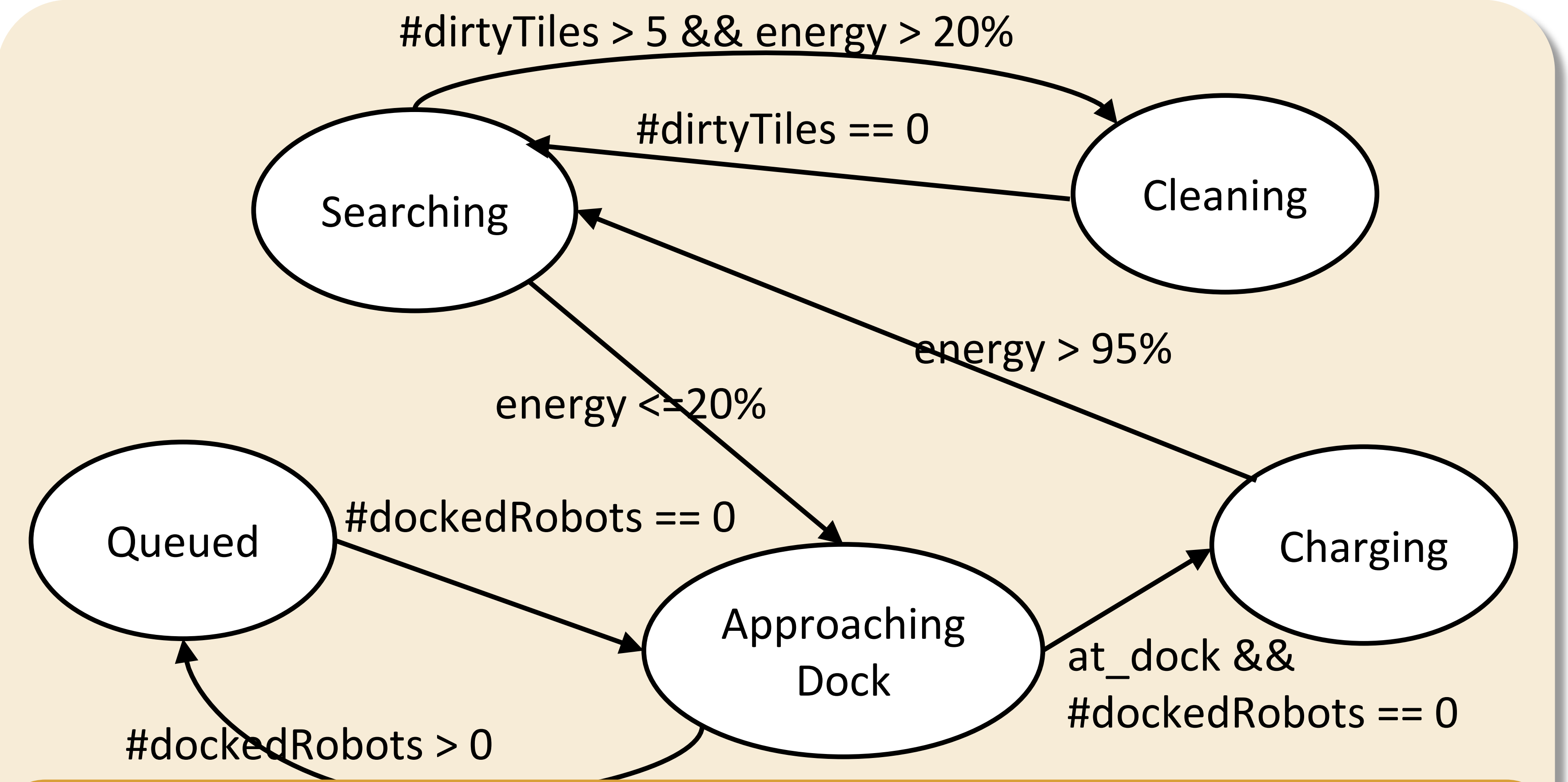


Illustration on




```
component Robot features Dockable, Cleaner {
  position: IPosition
  dirtinessMap: IMap
  targetPosition: IPosition
  assignedDockingStationsPosition: IPosition
  …
  process move in mode Cleaning, Searching {
    inputKnowledge =
              [position , targetPosition, dirtinessMap ]
    outputKnowledge = [position, dirtinessMap]
    function = {
        position ← move (targetPosition)
        dirtinessMap ← update(position, dirtinessMap)
    }
    scheduling = periodic(100 ms)
  }
  …
}
```



Self-adaptation mechanism #1: mode switching
(Via mode-state machines attached to components)

Illustration on Cleaning Robots

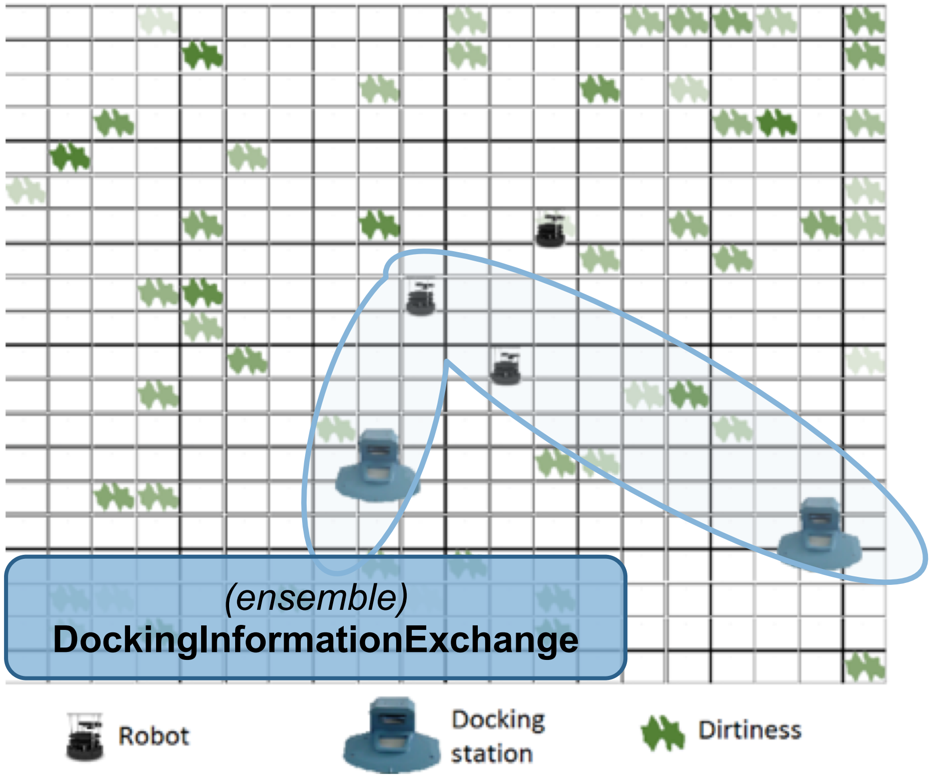


Illustration on Cleaning Robots

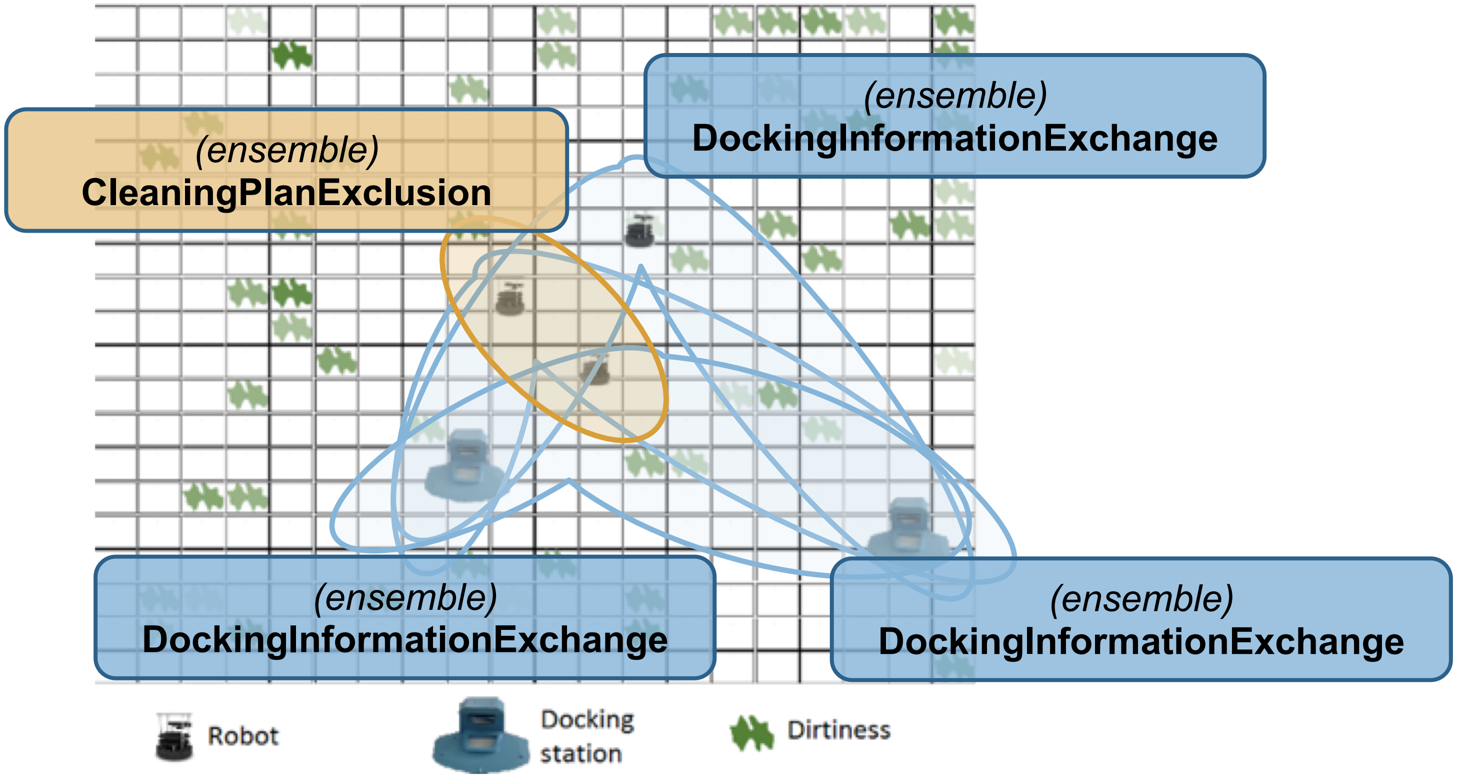
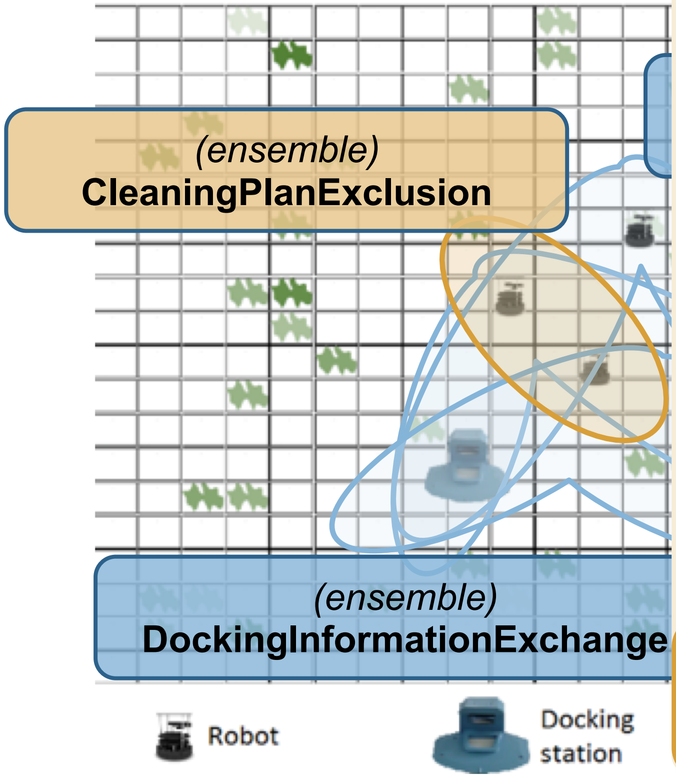


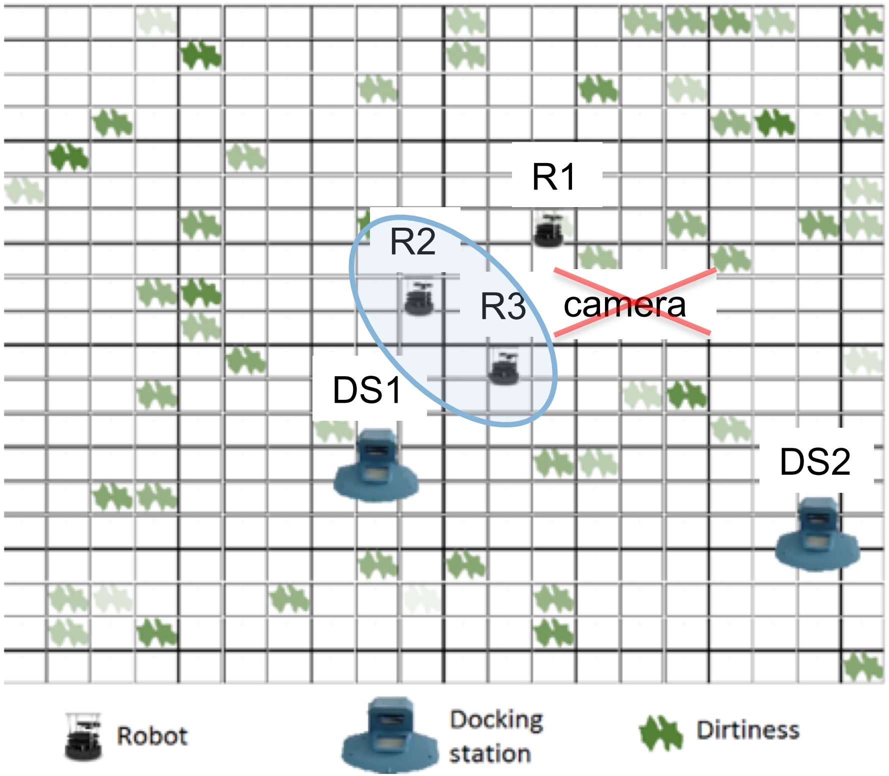
Illustration on Cleaning Robots




```
ensemble DockingInformationExchange = {
    coordinator = Dock
    member = Dockable
    membership = {
        coordinator.dockedRobots.size() <= 3
    }
    knowledge_exchange {
        coordinator.dockedRobots ← member.id
        member.assignedDockingStationPosition
            ← coordinator.position
    }
    scheduling = periodic(1000 ms)
}
```

Self-adaptation mechanism #2: ensembles

Homeostatic mechanism #1: Collaborative Sensing



Situation:

A robot's camera is broken → it cannot detect which tiles are dirty any more

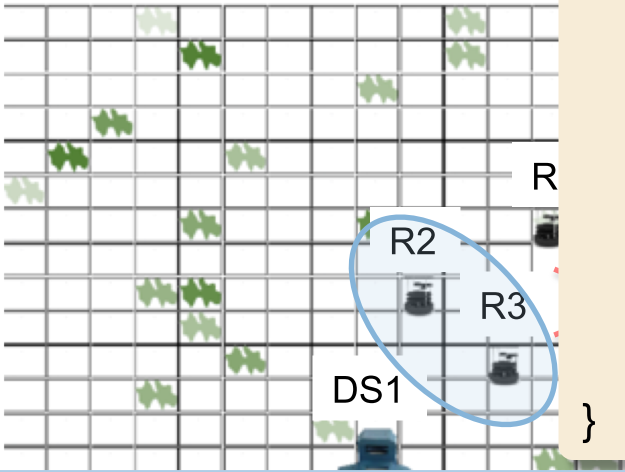
Solution:

Extend the self-adaptation mechanism of "ensembles" by creating a new ensemble that will copy the dirtinessMap of nearby robots

Available ensembles:

- CleaningPlanExclusion
- DockingInformationExchange
- DirtinessMapExchange

Homeostatic
Collaborative

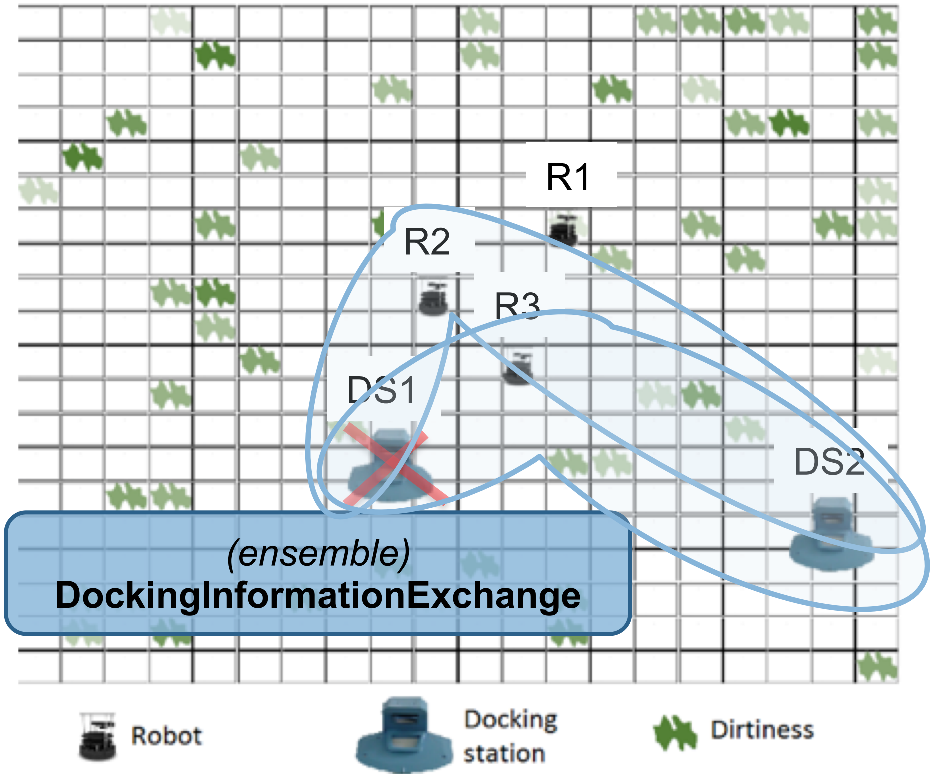



```
ensemble DirtinessMapExchange = {
    coordinator = DirtinessMapRole
    member = DirtinessMapRole
    membership = {
        close(coordinator.position, member.position)
          and obsolete(coordinator.dirtinessMap)
    }
    knowledge_exchange {
        coordinator.dirtinessMap ←
          member.dirtinessMap
    }
    scheduling = periodic(1000 ms)
}
```

How to create such an ensemble (one way):

- Store all data from all components
- Identify correlations between data series (e.g. when positions of two robots are close, their dirtiness maps are “close” as well)
- Translate correlations to ensemble specifications

Homeostatic mechanism #2: Faulty Component Isolation



Situation:
A docking station cannot charge docked robots anymore → a robot may still queue at a faulty docking station

Solution:
Exclude DS1 from being coordinator of one of the instances of the ensemble (to isolate the problem)

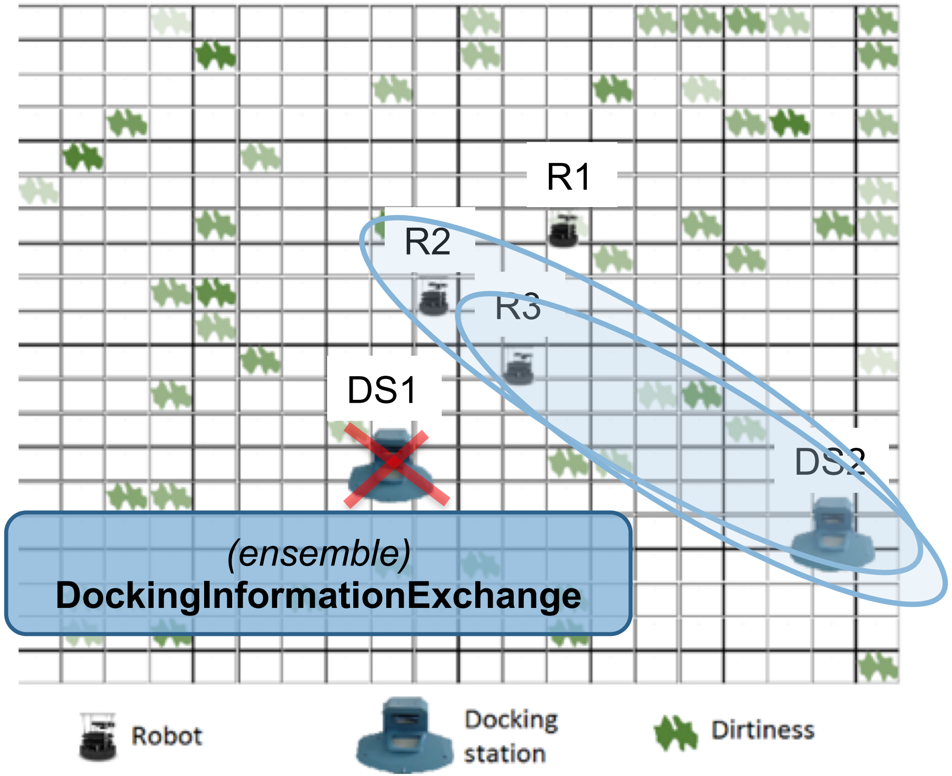
Robot's roles:

- Dockable
- Cleaner

Docking station's roles:

- ~~Dock~~

Homeostatic mechanism #2: Faulty Component Isolation



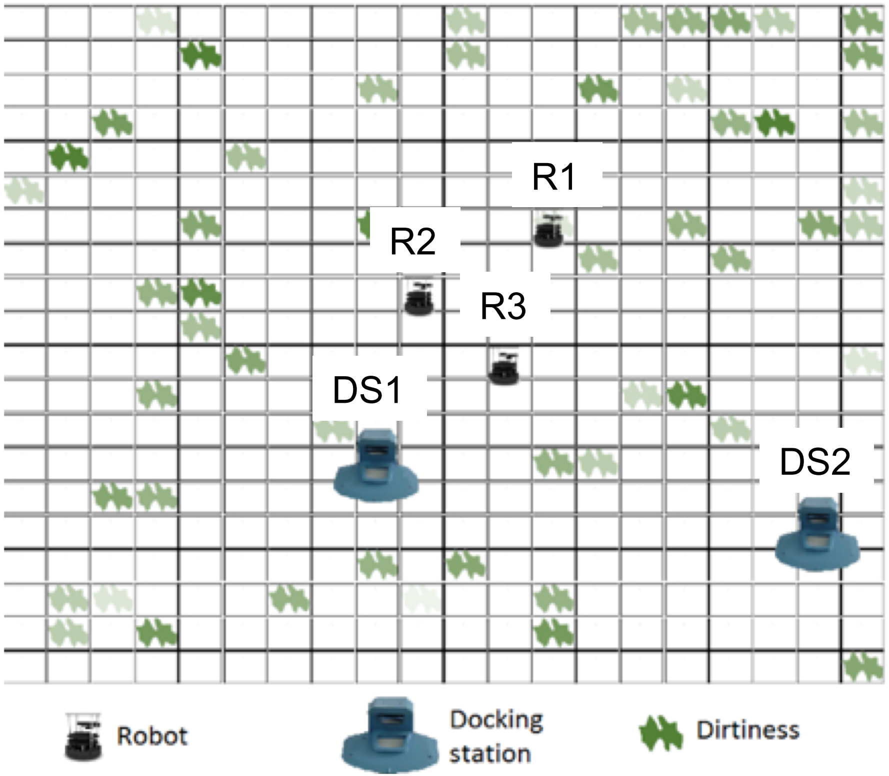
Situation:
A docking station cannot charge docked robots anymore → a robot may still queue at a faulty docking station

Solution:
Exclude DS1 from being coordinator of one of the instances of the ensemble (to isolate the problem)

Robot's roles:
- Dockable
- Cleaner

Docking station's roles:
- ~~Dock~~

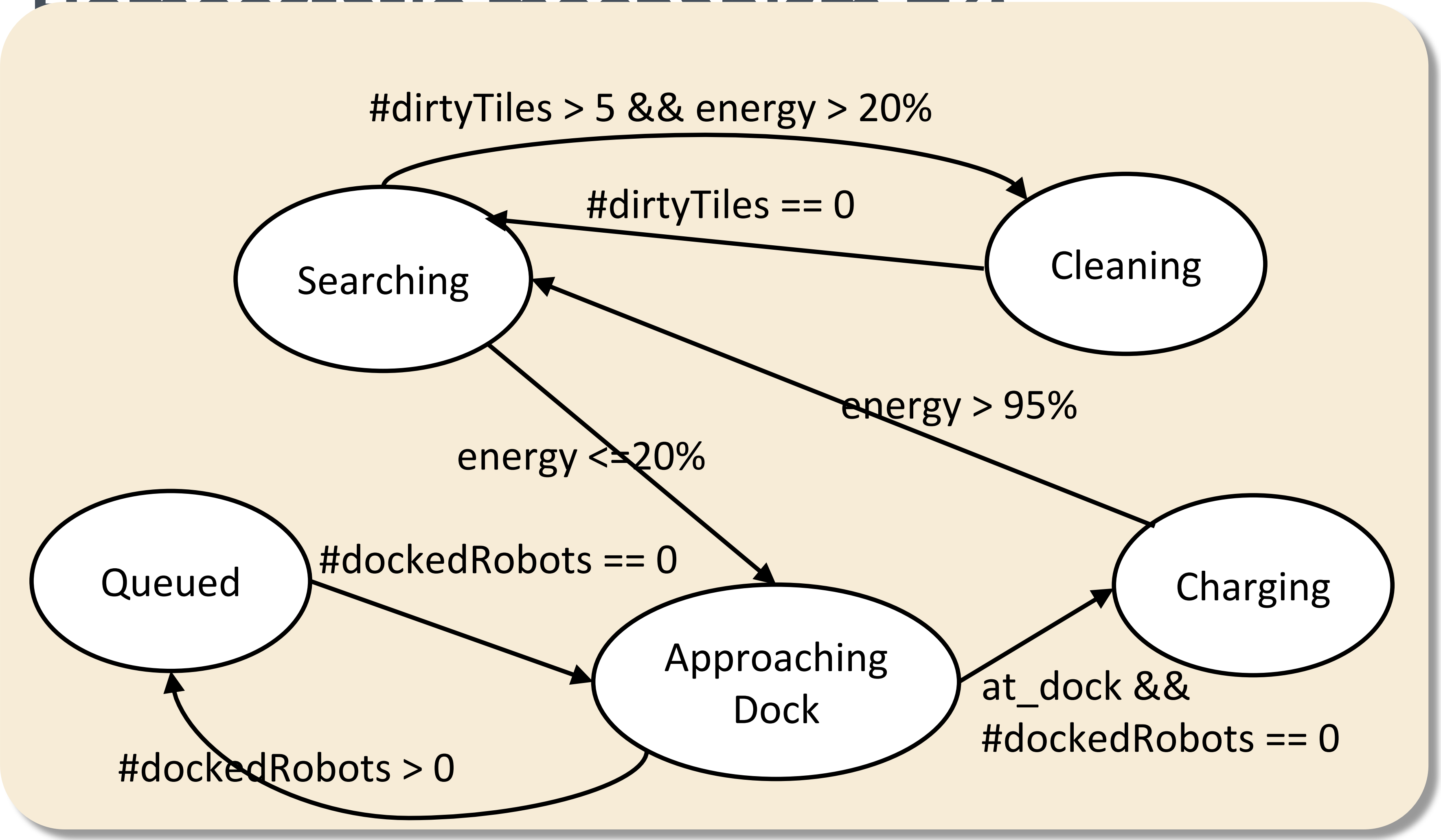
Homeostatic mechanism #3: Enhancing Mode Switching



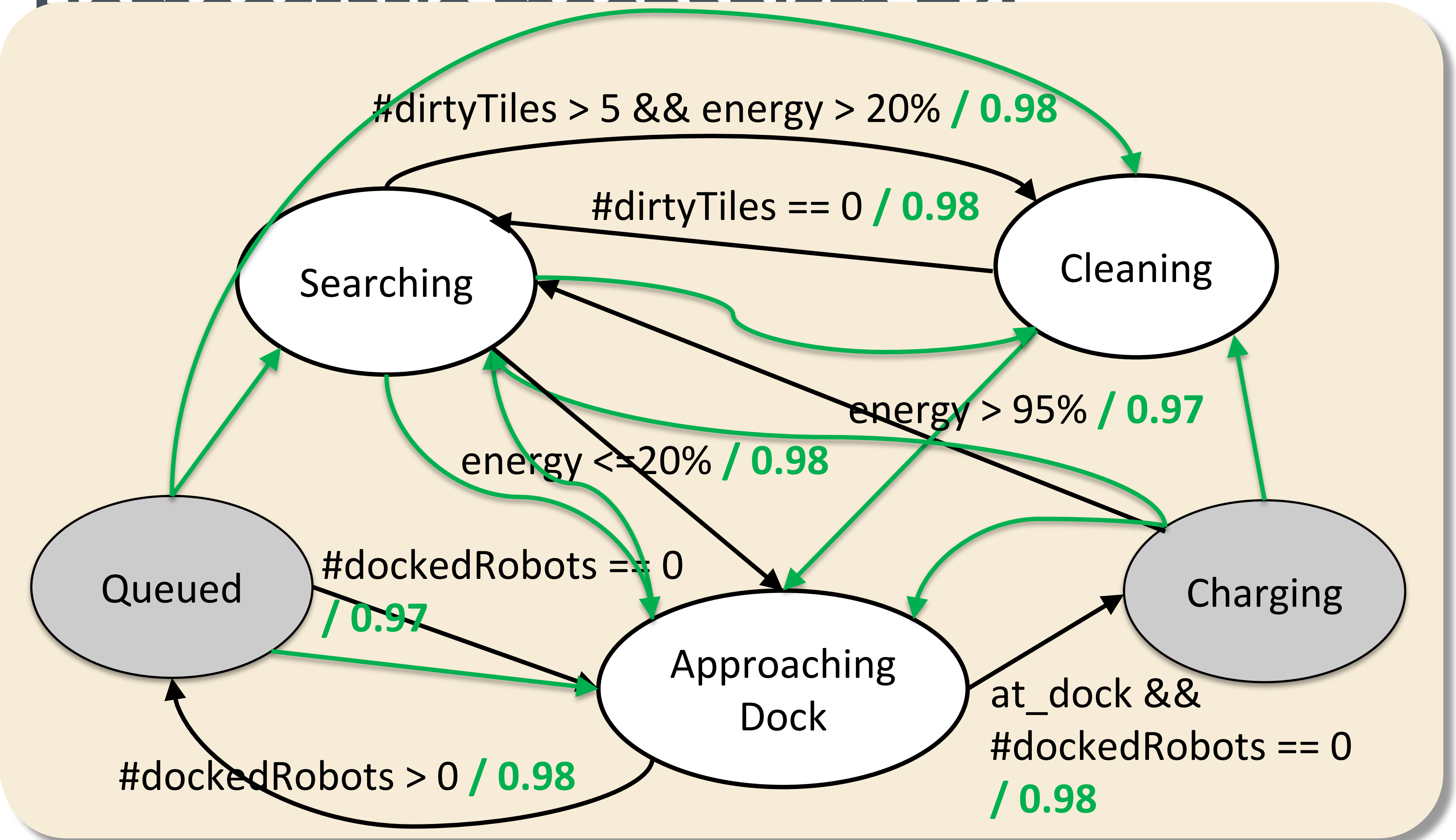
Situation:
Far more robots than docking stations → increased charging time because of queuing time

Solution:
Change the mode-state machine of robots to allow them to “break the regularity” in which robots go to recharge

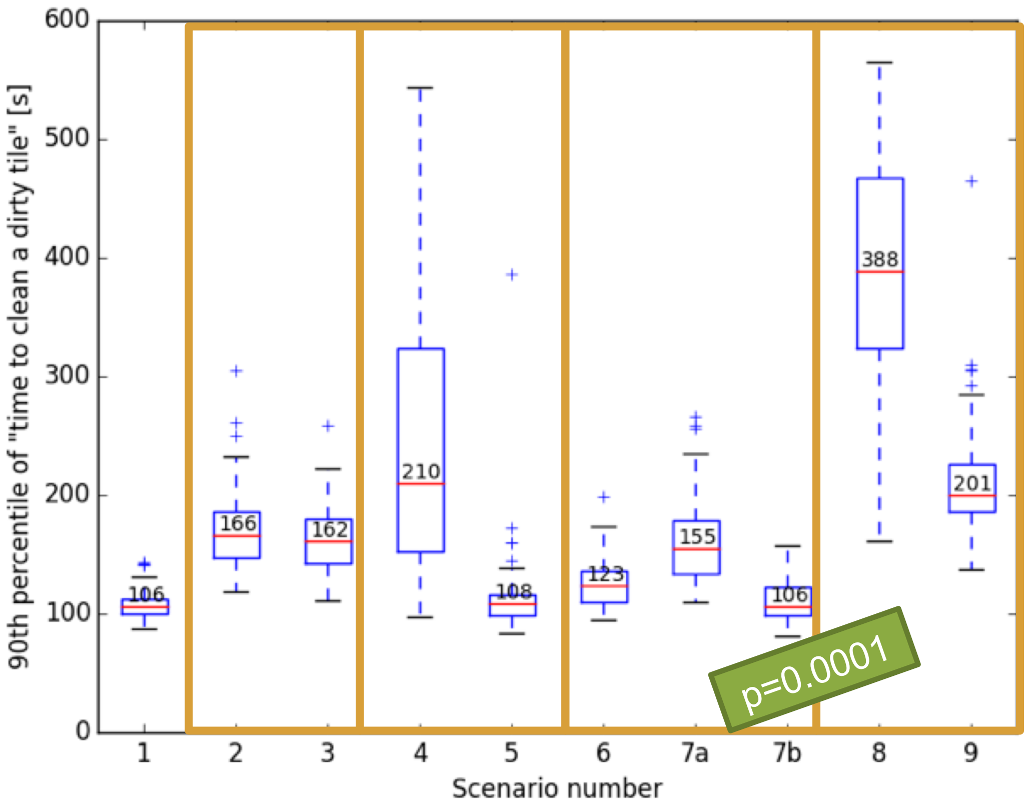
Homeostatic mechanism #7:



Homeostatic mechanism #7:



Experiments



1	-	-
2	A robot's dirtiness sensor malfunctions	-
3	A robot's dirtiness sensor malfunctions	#1
4	A docking station emits wrong availability data	-
5	A docking station emits wrong availability data	#2
6	Too many robots w.r.t. docking stations	-
7	Too many robots w.r.t. docking stations	#3
8	All above	-
9	All above	all

What we learned

Introducing uncertainty to the system can indeed help (esp. considering the results of enhanced mode switching)

Homeostatic mechanisms are specific to adaptation strategies -> hard to generalize

Expert domain knowledge is needed to specify and implement the mechanisms inspirational moment #2

... a balance where R&D teams build part of the functionality and **set guardrails**, and where smart systems **experiment** and adjust their responses and behaviors **autonomously**

Bosch and Olsson, “Data-Driven Continuous Evolution of Smart Systems”, SEAMS 2016

(My) definition of different experimentation types

- Empirical experimentation

e.g. running a controlled experiment with students

- Online experimentation

e.g. A/B test at Google, Facebook, Netflix, …

- Continuous experimentation

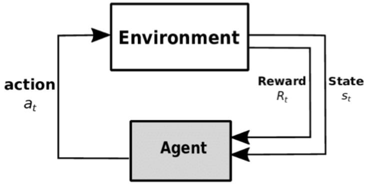
e.g. bandit algorithms

- Automated experimentation

Bosch and Olsson's vision

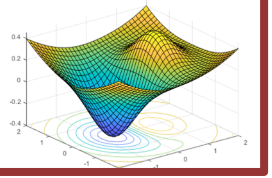
How to achieve "automated experimentation"?

- Self-adaptive system as **reinforcement learning** system?

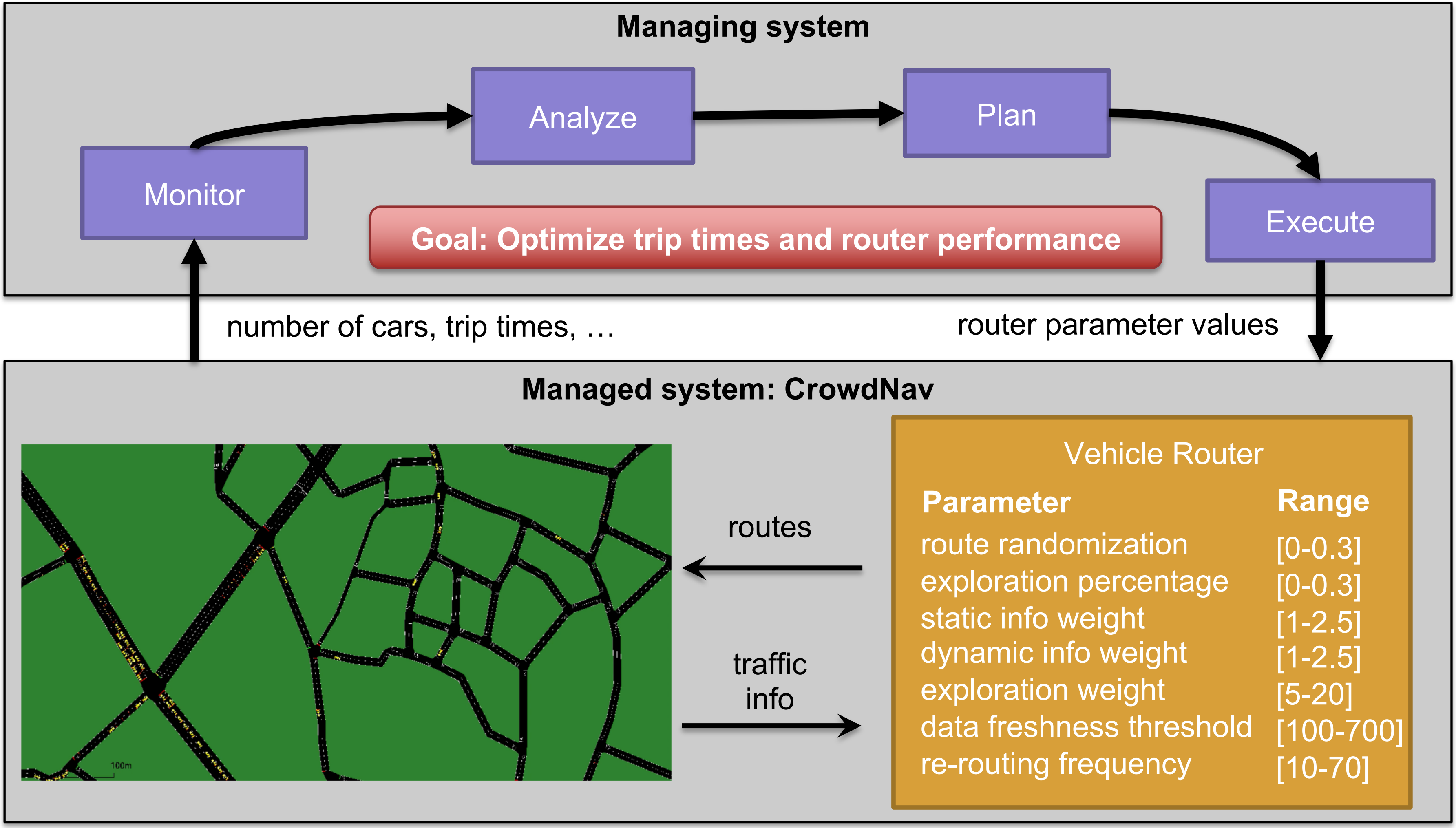


- Self-adaptive system that formulates and (statistically) tests **hypotheses** at runtime?

- Self-adaptive system with the ability to **compare and use optimizers** at will?

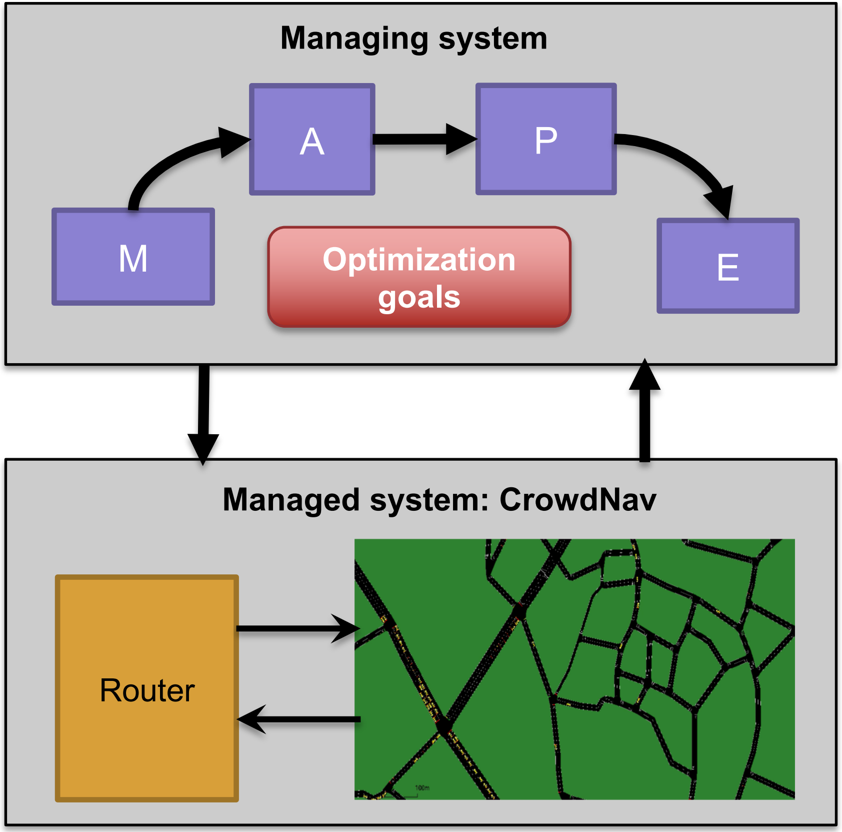


- What about **cost vs benefit** of automated experimentation?


Managing system
Monitor
Analyze
Plan
Execute
Goal: Optimize trip times and router performance
number of cars, trip times, …
router parameter values
Managed system: CrowdNav
100m
routes
traffic info
Vehicle Router
Parameter
Range
route randomization
[0-0.3]
exploration percentage
[0-0.3]
static info weight
[1-2.5]
dynamic info weight
[1-2.5]
exploration weight
[5-20]
data freshness threshold
[100-700]
re-routing frequency
[10-70]

The case of Optimizing CrowdNav

1. There are different environment situations e.g. high/low/normal traffic, blocked streets, …
2. The managed system can have different **configurations** → valuations of router parameters
3. An **optimal configuration** minimizes trip times and minimizes the time spend in routing
4. It is unlikely that an **optimal configuration** will work **in all situations**
5. It is **difficult to enumerate all possible situations**



One way of optimizing CrowdNav

Specify all **possible situations** and their optimal **configurations**

- Enumerate them
- Specify permissible situations via a model (e.g. DTMC)

Design-time

Use **rules** to apply situation-optimal configurations

Run-time

+

-> easy to encode & interpret

–

-> difficult to derive (extensive simulations? detailed system model?)

-> difficult to extend (new situations? new configurations?)

Our way of optimizing CrowdNav

- Specify system **input and output parameters** & optimization **goals**
- Specify **context** (environment) **parameters**

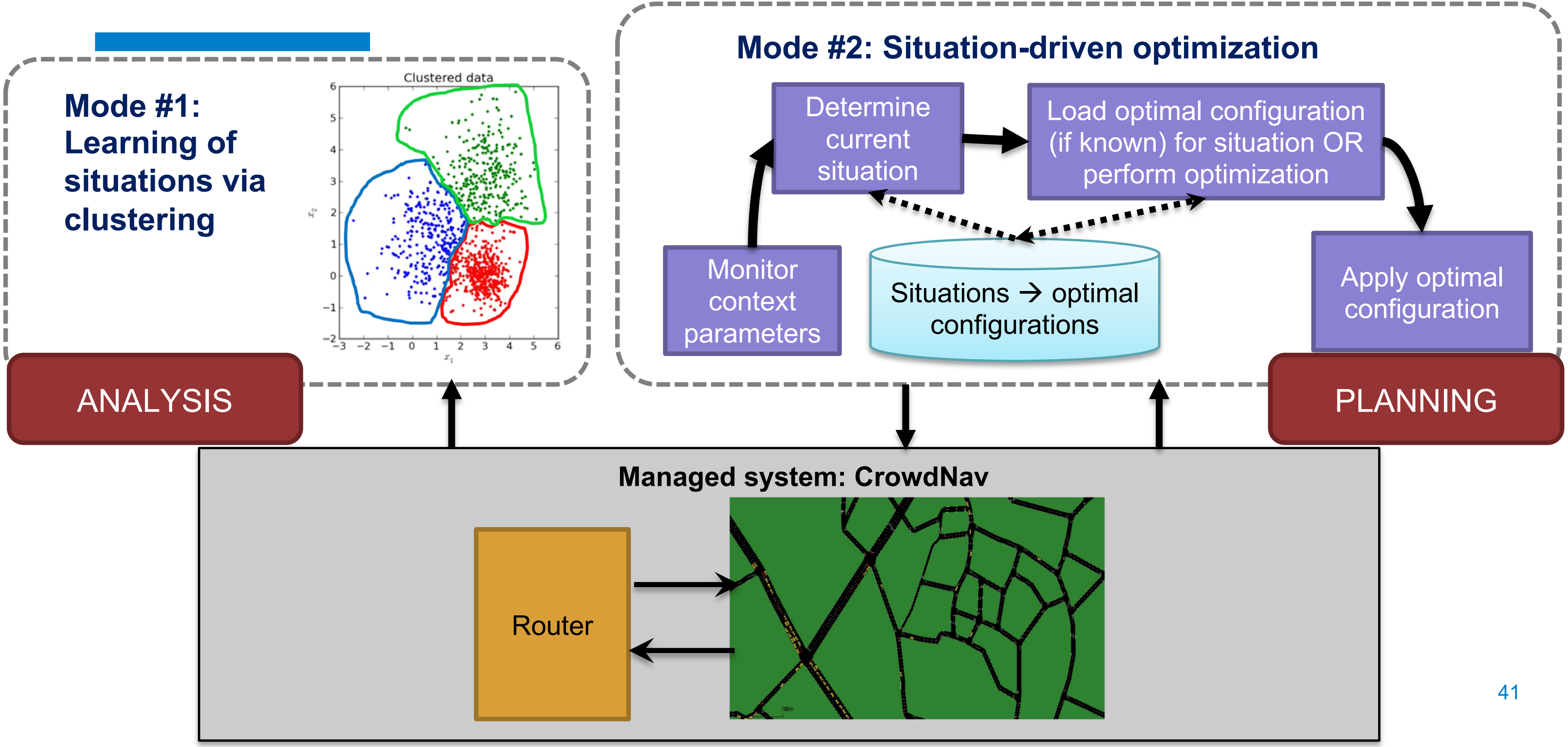
Design-time

- Identify **situations** via the effect of context parameters on the outputs
- Use an **optimization strategy** to identify the optimal configuration for each identified situation at runtime

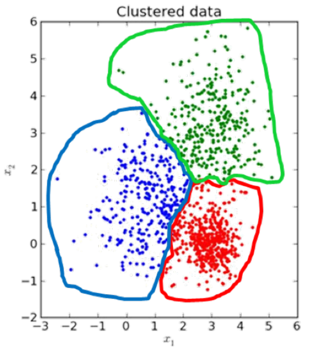
Run-time

"Planning as Optimization"

Planning as optimization: Overview



Mode #1: Learning of situations via clustering



Goal of this mode

Determine **situations** via grouping together environment states based on the effect they have on system outputs

Assumptions

Each context parameter has a number of states (e.g. ranges)

number_of_cars in [0,100], [101, 200], [201, 300], [301, ∞)

percentage_of_blocked_streets in [0,25], [26-50], [51-75], [76-100]

→ All the possible **environment states** is the Cartesian product of the states of all context parameters

Method

Continuously collect values of system outputs and environment states

Compute (statistical) features for each state-dataset

e.g. mean, variance, 95^{th} percentile, …

Use clustering at runtime to group environment states in **situations**

Evaluation of learning of situations via clustering

number_of_cars in [100-150], (151-200], ..., (751-800] context parameter

trip_overhead: normalized trip duration output parameter

100-150 | 151-200 | . . . | 701-750 | 751-800

state-datasets

100-150	151-200	701-750	751-800
average	average	average	average
median	median	median	median
75th percentile	75th percentile	75th percentile	75th percentile
90th percentile	90th percentile	90th percentile	90th percentile
variance	variance	variance	variance
standard deviation	standard deviation	standard deviation	standard deviation

features of state-datasets

k-means algorithm with *k* in 2..9 → **Silhouette** method to find best *k*

Evaluation of learning of situations vi

num parameter

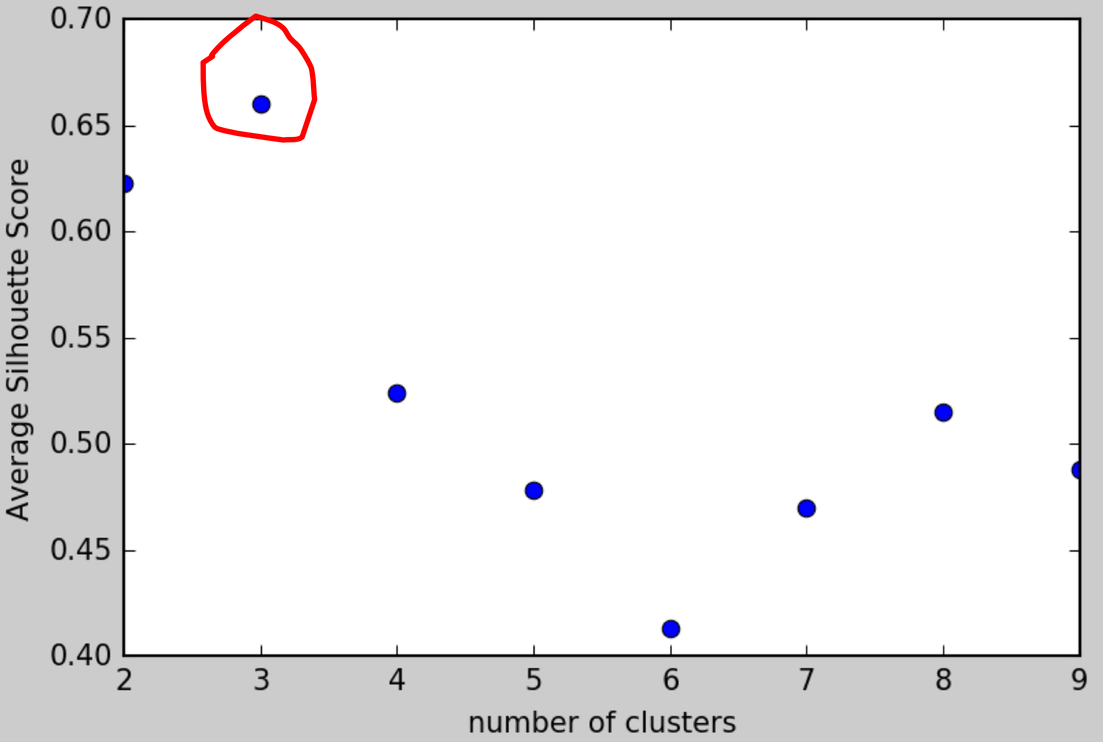
trip parameter

100-1

average
median
75th p
90th p
varianc
standa

datasets

es of
datasets



1st cluster: *number of cars* in 0..500 → "low traffic"
2nd cluster: *number of cars* in 501..700 → "medium traffic"
3rd cluster: *number of cars* in 701..800 → "high traffic"

k-means algorithm with *k* in 2..9 → **Silhouette** method to find best *k*

Mode #2: Situation-driven optimization

Goal of this mode

Determine **optimal configurations** via online search in the space of possible configurations for each situation

Assumptions

The optimization process can update the system configuration on the fly

The optimization process is not interrupted once started

Question

Which optimization algorithm guides the optimization process best?

→ Many options: linear programming, genetic algorithms, local search, combinatorial optimization, stochastic optimization, …
→ Depends on the managed system and the characteristics of the situations that it resides in

CrowdNav as a numeric optimization problem

I.

Black-box
(no known model that relates inputs to outputs)

II.

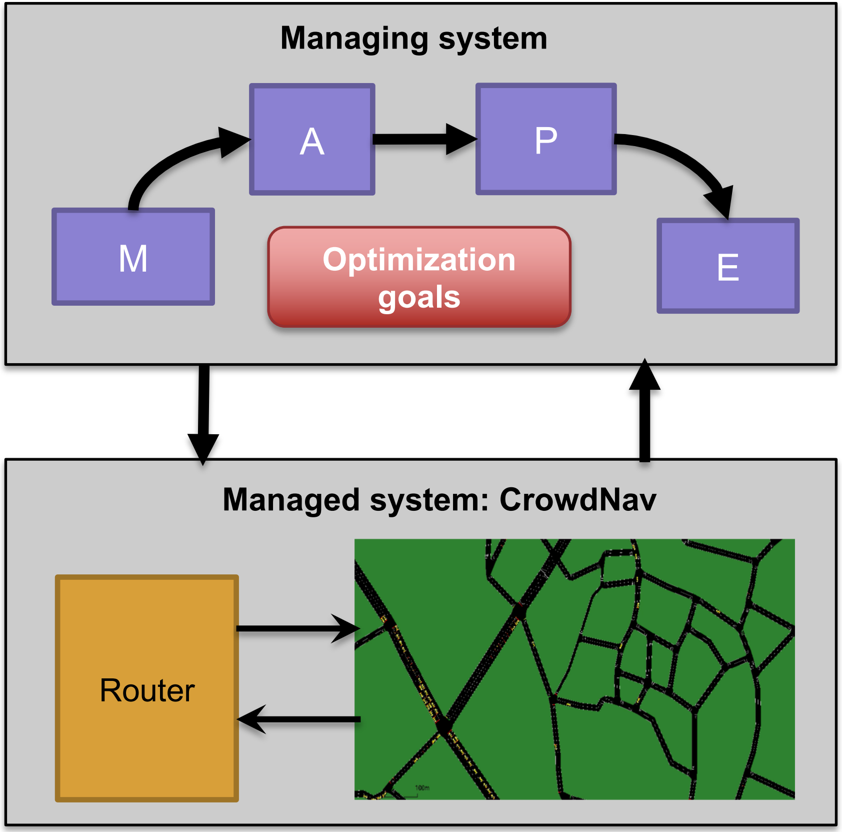
High dimensional
(large space of configurations)

III.

Expensive
(many samples needed to evaluate a configuration)

IV.

Multi-objective
(minimize both overhead and router performance)



Optimization algorithms considered

Bayesian optimization

Given a number of steps, at each step, try a configuration, collect output values and fit a regression model (e.g. Gaussian process)

Good for expensive black-box optimization of continuous spaces

Non-dominated Sorting Genetic Algorithm II (NSGA-II)

A solution (configuration) is modeled as a chromosome

Mutation, crossover operators

Fitness function evaluates a configuration and guides search

Good for multi-objective evolutionary search

Novelty Search

Similar to NSGA-II, but fitness measured based on novelty metric

Good for not being “stuck” in local optima

Evaluation of situation-driven optimization (on CrowdNav)

Which optimization method is best for CrowdNav

…for each situation?

…across all situations?

We compared the 3 methods (+ random search) based on

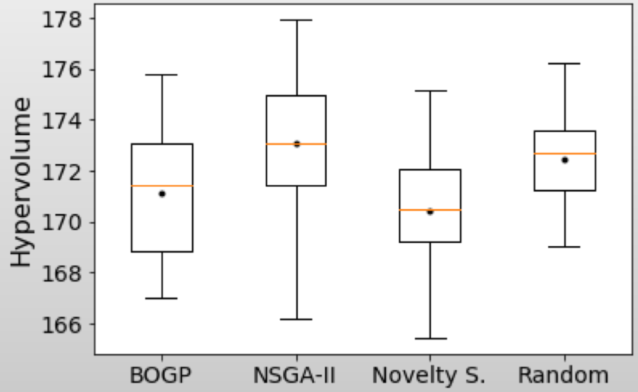
- **solution quality**: how well the two objectives are minimized
- **convergence**: how quickly the search stagnates
- **overhead**: memory and processor usage needed

…with a 100-step budget per optimization run

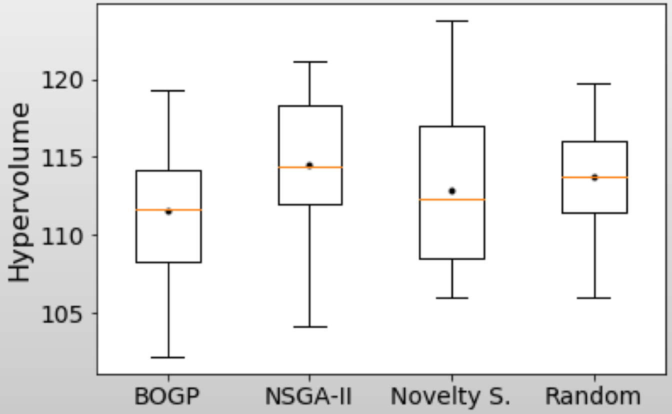
…with 30 replications or each run to obtain statistical validity

…for each of the three identified situations

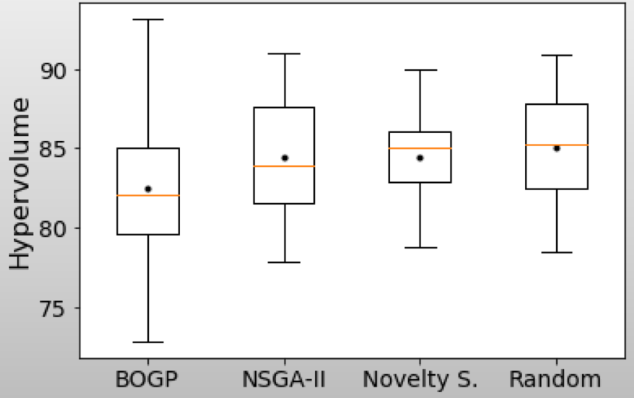
Results: solution quality



Low traffic

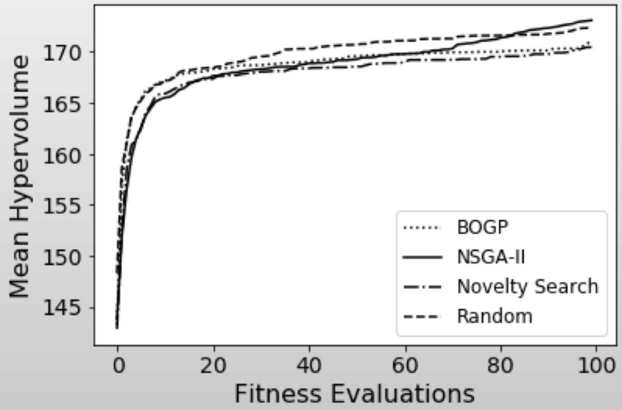


Medium traffic

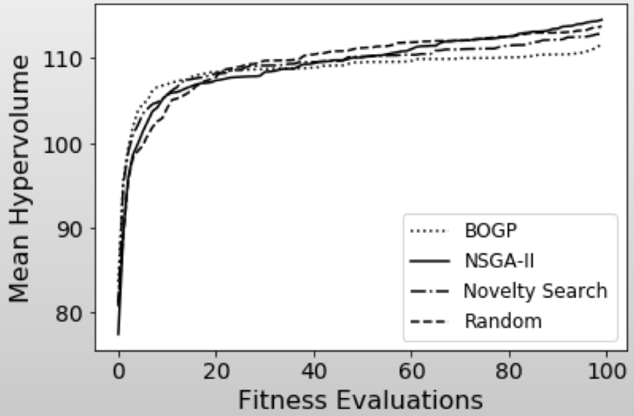


High traffic

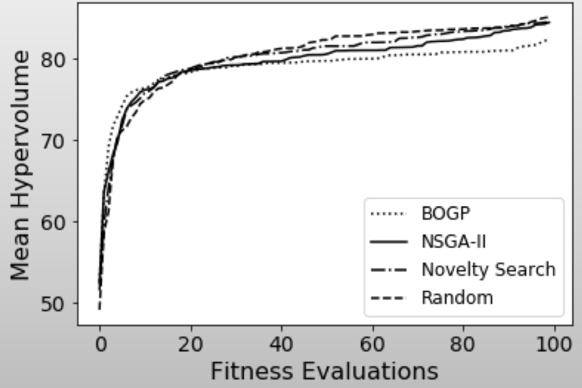
Results: convergence



Low traffic



Medium traffic



High traffic

Answering the research questions

Which optimization method is best for CrowdNav
...for each situation?
...across all situations?

Pareto-optimal configurations are spread all over the search space → Many local minima!

Based on solution quality

NSGA-II performs better in “low” and “medium” traffic
Random search performs better in “high” traffic

Based on convergence

Bayesian optimization performs (slightly) better

Based on overhead

They are all equally good

What we learned

Clustering is a viable option for identifying situations (but needs to be done continuously)

Challenge: Automated comparison of optimizers (different evaluation criteria, unclear evaluation horizon)

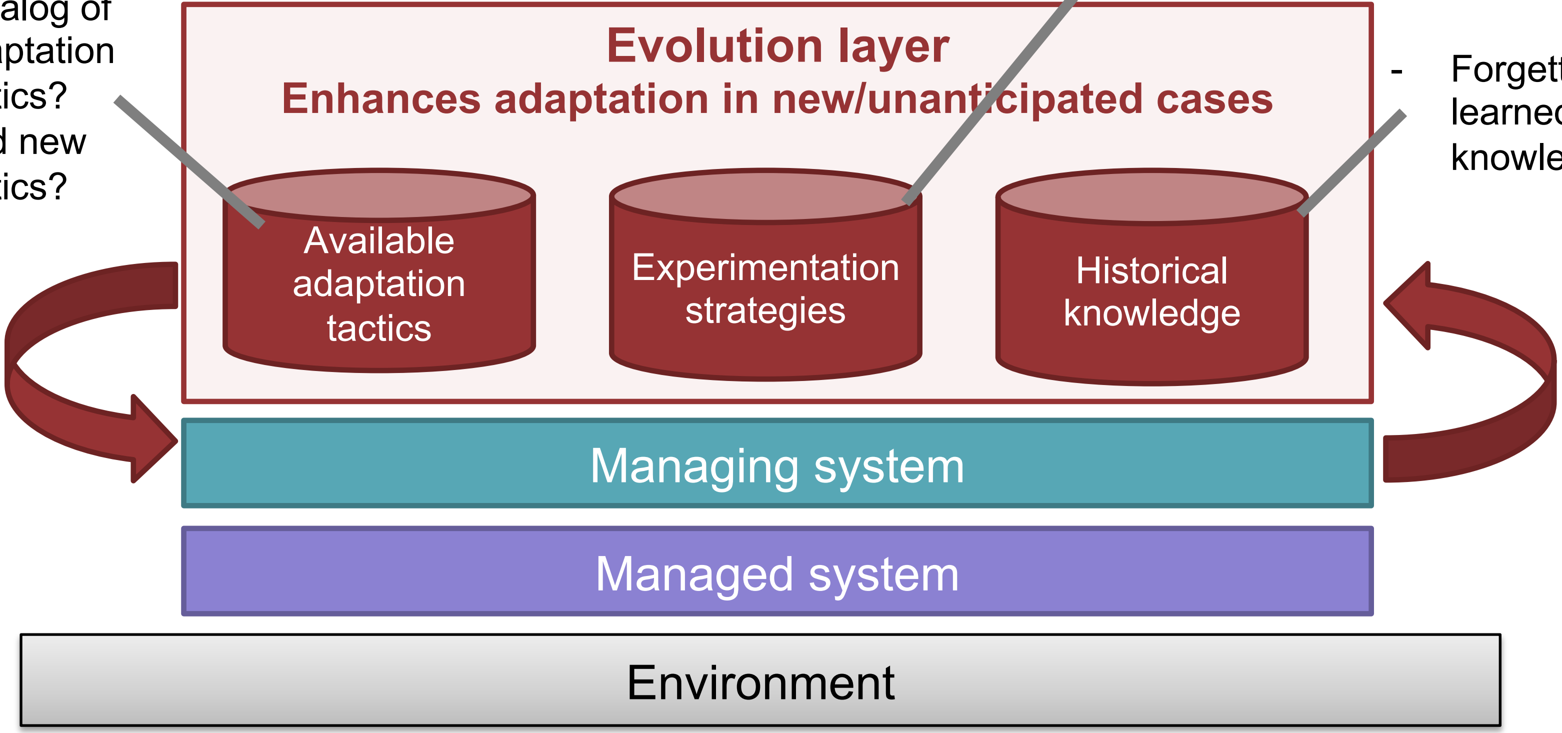
Challenge: Lifecycle management of optimizers (they need to be started, paused, stopped, etc.)

Vision: Self-evolution

- When to start a new experimentation round (novelty detection)?
- Which strategy to use?
- Guarantees?

- Catalog of adaptation tactics?
- Add new tactics?

- Forgetting learned knowledge?



(Some) research directions

What about using algorithms that consider context changes (e.g. contextual bandits, contextual genetic algorithms)?

How to deal with the tradeoff between increased complexity of the system and its increased ability to deal with unanticipated situations (cost-benefit analysis)?

How to devise a method for building (self-evolving) self-adaptive systems?

References

- Ilias Gerostathopoulos, Dominik Skoda, Frantisek Plasil, Tomas Bures, Alessia Knauss
  Architectural Homeostasis in Self-Adaptive Software-Intensive Cyber-Physical Systems
  10th European Conference on Software Architecture (ECSA'16)
- Ilias Gerostathopoulos, Dominik Skoda, Frantisek Plasil, Tomas Bures, Alessia Knauss
  Tuning Self-Adaptation in Cyber-Physical Systems through Architectural Homeostasis
  Elsevier's Journal of Systems and Software (JSS), Volume 148, February 2019, Pages 37-55
- Erik Fredericks, Ilias Gerostathopoulos, Christian Krupitzer, Thomas Vogel
  Planning as Optimization: Dynamically Discovering Optimal Configurations for Runtime Situations *13th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2019)*
- Ilias Gerostathopoulos, Frantisek Plasil, Christian Prehofer, Janek Thomas, Bernd Bischl
  Automated Online Experiment-Driven Adaptation–Mechanics and Cost Aspects
  IEEE Access

and other available at https://iliasger.github.io/publications/